ASSIGNMENT 1

INTRODUCTION TO NLP

SRIHARSHITHA BONDUGULA

2018111013 30/1/2021

Creating a Language model

First part of the language modelling is to clean the text. As a part of cleaning the text, following operations are done on the sentences using regex.

Text cleaning:

1) Convert the sentences to lower case.
2) Remove punctuations from the sentences. “Dr.” now becomes “dr” and “B.P.” becomes “bp”
3) Consider only alphabets. (Removes numbers and other languages)
4) Remove the lines with the “_” string. (To remove the sentences with POS tags)

Kneyser-ney’s model of smoothing:

I have implemented the following recursion formula, taking n=4 (as the model we are building is 4-gram LM).

$$P_{\mathrm{KN}}(w_i|w_{i-n+1:i-1}) = \frac{\max(c_{KN}(w_{i-n+1:i}) - d, 0)}{\sum_v c_{KN}(w_{i-n+1:i-1}\ v)} + \lambda(w_{i-n+1:i-1}) P_{KN}(w_i|w_{i-n+2:i-1})$$

Lambda for a bigram is:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}|$$

The same approach is used to find the lambda for other grams.

$c_{KN}(.)$ is count(.) for 4-grams and continuation count for all other lower order grams (trigram, bigram and unigram) where the continuation count is the number of unique single word contexts for ·.

The termination condition for recursion is when P_{KN} of a unigram is called. It is as follows,

$$P_{\text{KN}}(w) = \frac{\max(c_{KN}(w) - d, 0)}{\sum_{w'} c_{KN}(w')} + \lambda(\epsilon)\frac{1}{V}$$

The Lambda of epsilon is 'd'.
Value of d is taken as 0.75.

Code snippet:

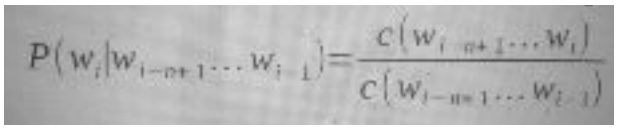
```
def calc_kneyser_probability(sentence):
  length = len(sentence.split())
  if (length >=2): return first_term(sentence) + kneyser_lambda(sentence)*calc_kneyser_probability(gen_post_string(sentence))
  else: return first_term(sentence) + (kneyser_lambda(sentence)/len(unigrams))
```

Witten bell's model of smoothing:

I have implemented the following recursion formula taking n=4.

$$p_{\text{WB}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\, p_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}})\, p_{\text{WB}}(w_i|w_{i-n+2}^{i-1})$$

The formula for p_{ML} is as follows,



Formula used to compute lambda is as follows,

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})}$$

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1}w_i) > 0\}|$$

Recursion terminates when p_{WB}(unigram) is called.

Code snippet:

```
def calc_witten_probability(sentence):
    length = len(sentence.split())
    if (length >=2): return witten_lambda(sentence)*mle_probability(sentence) + (1-witten_lambda(sentence))*calc_witten_probability(gen_post_string(sentence))
    else: return witten_lambda(sentence)*mle_probability(sentence) + (1-witten_lambda(sentence))*(1/(2*len(unigrams)))
```

Computing perplexity - to evaluate the language model:

I have used the following formula to compute the perplexity of a sentence W = $w_1w_2...w_N$

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

$P(w_1w_2...w_N)$ can be expanded using chain rule based on the LM we are building (4-gram in our case)..

Code snippet:

```
perplexity = probability**(-1/len(input.split()))
```

Observations:

Splitting the corpus into test and train:

Random 1000 lines from the corpus are taken as the test set and the model is trained on the other lines, i.e the training set.

```
for x in file:
    r1 = random.randint(1, 10)
    if(r1%2==0): f_train.write(x)
    elif(r1%2!=0 and counter<1000):
        f_test.write(x)
        counter+=1
    else: f_train.write(x)
```

Average perplexities of train sentences:

	Kneyser-Ney	Witten-bell
Health Corpus	17.5307580665028	10.8926669926752935
Technical Corpus	3.777256590678746	2.615631941089133

- Average perplexities for both the train corpuses are computed after training both the models on the same set.
- Average of only a few (500-1000) sentences from the train set is computed due to the time constraint.

- From the above table, the average perplexities of training sets of both the corpuses from Kneyser-ney model are higher than the average perplexities from Witten-bell model. Thus, the probabilities given by the Kneyser-ney model must have been lesser than those given by the Witten-bell model.
- This tells that Witten-bell is more conservative and deducts lesser value from probabilities of seen sentences, i.e the sentences in the training set. Whereas the penalty imposed by Kneyser-ney on seen sentences is higher.

Average perplexities of test sentences:

	Kneyser-Ney	Witten-bell
Health Corpus	101.74828224499177	331.2303428948926
Technical Corpus	48.40029692335503	134.526656369984

- Average perplexities for both the test corpuses are computed after training both the models on a separate training set.
- From the table above, the average perplexities of both the corpuses given by Witten-bell are higher than Kneyser-Ney. This means that the average probabilities for both the corpuses given by Witten-bell are lesser than those given by Kneyser-Ney's. This proves that Witten-bell is conservative and takes less probabilities from seen sentences and hence gives lesser probabilities to unseen sentences. Comparatively, penalty on seen sentences is more when the Kneyser-ney model is used and hence probabilities for them are high.
- The lower the perplexity on the test corpus, better is the model.
- Hence, my conclusion is that the Kneyser-Ney model is better than the Witten-bell model.

RESULTS FOR A SEEN SENTENCE

For the sentence - "drink plenty of water" - from the health domain corpus, after both the models are trained on the corpus, the probability given by Kneyser-Ney is 0.102 which is **lower** than the one given by Witten-Bell, i.e, 0.172.

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py k ./Health_English.txt
Enter the sentence: drink plenty of water
Probability of the sentence is  0.10295910732224445
Perplexity of the sentence is  1.4606826539558615
```

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py w ./Health_English.txt
Enter the sentence: drink plenty of water
Probability of the sentence is  0.17203588542540582
Perplexity of the sentence is  1.340901342222623
```

RESULTS FOR A PARTIALLY SEEN SENTENCE - seen phrases + unseen words

For the sentence - “drink plenty of juice” - not from the health domain corpus, after both the models are trained on the corpus, the probability given by Kneyser-Ney is 6.11×10^{-5} which is **higher** than the one given by Witten-Bell, i.e, 7.44×10^{-6} .

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py k ./Health_English.txt
Enter the sentence: drink plenty of juice
Probability of the sentence is  6.112000976708643e-05
Perplexity of the sentence is  5.038517417179293
```

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py w ./Health_English.txt
Enter the sentence: drink plenty of juice
Probability of the sentence is  7.440107430710179e-06
Perplexity of the sentence is  7.157095280669682
```

RESULTS FOR AN UNSEEN SENTENCE

For the sentence - “all that glitters is not gold” - not from the health domain corpus, after both the models are trained on the corpus, the probability given by Kneyser-Ney is 1.90×10^{-15} which is **quite higher** than the one given by Witten-Bell, i.e, 3.28×10^{-19} .

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py k ./Health_English.txt
Enter the sentence: all that glitter is not gold
Probability of the sentence is  1.901161275809059e-15
Perplexity of the sentence is  69.20263481127145
```

```
dolly@sriharshitha:~/Acads/Sem6/NLP/Assignment1$ python3 language_model.py w ./Health_English.txt
Enter the sentence: all that glitters is not gold
Probability of the sentence is  3.2889453804033326e-19
Perplexity of the sentence is  204.34683583038097
```

Hence,

1) Both the models do not give 0 probability for any unseen sentence.
2) The table below compares the probabilities and perplexities computed by both the models for different types of input sentences.

	Kneyser-ney probability	Witten-bell probability
Seen	Comparatively lower	Higher
Partially seen	Higher	Comparatively lower
Completely unseen	Quite higher	Lower

	Kneyser-ney perplexity	Witten-bell perplexity
Seen	Comparatively higher	Lower
Partially seen	Lower	Comparatively higher
Completely unseen	Lower	Quite higher

3) Witten-bell is conservative.
4) When evaluated based on average perplexity, Kneyser-ney is a better model than Witten-bell.

References:

https://web.stanford.edu/~jurafsky/slp3/3.pdf
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-10-98.pdf?from=https%3A%2F%2Fresearch.microsoft.com%2F%7Ejoshuago%2Ftr-10-98.pdf